RADC-TR-90-92
Final Technical Report
May 1990

AD-A224 489

IMPLEMENTATION ISSUES IN MULTILEVEL SECURITY FOR OBJECT-ORIENTED DATABASES

George Mason University

Sushil Jajodia, Boris Kogan

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-92 has been reviewed and is approved for publication.

APPROVED: Joseph V Giordano.

JOSEPH V. GIORDANO
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Igor G. Plonisch

IGOR G. PLONISCH
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1990	Final Jun 89 to Dec 89

4. TITLE AND SUBTITLE
IMPLEMENTATION ISSUES IN MULTILEVEL SECURITY FOR OBJECT-ORIENTED DATABASES

5. FUNDING NUMBERS
C - F30602-88-D-0028
PE - 35167G
PR - 1068
TA - 01
WU - P4

6. AUTHOR(S)
Sushil Jajodia, Boris Kogan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
George Mason University
Dept of Information Systems & Systems Engineering
4400 University Drive
Fairfax VA 22030-4444

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Air Development Center (COTD)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RADC-TR-90-92

11. SUPPLEMENTARY NOTES
RADC Project Engineer: Joseph V. Giordano/COTD/(315) 330-2925

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report concentrates on implementation issues associated with the security model for object-oriented databases. The discussion of the model is conducted from the implementation point of view. Certain characteristics the model possesses, make its implementation a conceptually simple matter. Two alternative approaches to the subject of implementation are analyzed. Finally, the requirements the model places on the implementation of the object layer are explained in detail.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Object-oriented, database management systems, multilevel security, trusted systems	28
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

ACKNOWLEDGEMENTS

This work was partially supported by the U. S. Air Force, Rome Air Development Center through subcontract # RI-64155X of prime contract # F30602-88-D-0028, Task B-9-3622 with University of Dayton. We are indebted to RADC for making this work possible.

ABSTRACT

This report concentrates on implementation issues associated with the security model for object-oriented databases introduced by Jajodia and Kogan [4]. The discussion of the model is conducted from the implementation point of view. We argue that, due to certain characteristics that our model possesses, its implementation turns out to be a conceptually simple matter. We analyze two alternative approaches to the subject of implementation. Finally, we explicate the requirements that our model places on the implementation of the object layer.

1. Introduction.

This report concentrates on implementation issues associated with the security model for object-oriented databases introduced by Jajodia and Kogan [4]. The single most important point that we will try to make here is that the nature of our model is such that it already contains the recipes for implementation. In other words, it would be wise to consider the questions of the model and its implementation in a highly integrated way rather than to place them at related but separate conceptual levels.

Because of the preceding consideration we think it advisable to start this report with a brief but self-contained description of the model of [4].

The main motivation of our work in [4] was to construct a security model that would integrate in a very natural way with an object-oriented model rather than juxtapose with it. To that end we decided to move away from the traditional object-subject paradigm of Bell and LaPadula [2] for multilevel security. In its place we introduced a new paradigm whose main elements were *objects* (in the new, object-oriented sense) and *messages*.

In an object-oriented data model, the notion of object is rather different from the Bell-LaPadula notion of object. Whereas the latter is simply a file, or a record, or a field, the former can be regarded as a passive repository of data and, at the same time, as an active agent manipulating those data and communicating with other objects. It seems natural, therefore, that such objects be viewed as units of security. Perhaps the most important consequence of adopting such a view is that, due to the property of encapsulation,[†] information flow becomes explicit (in the form of message exchange among

[†] Encapsulation of objects — a fundamental property of object-oriented systems — means that only objects themselves can have direct access to their internal state (their attributes). For anyone

objects) and, therefore, easy to control. Thus, the notion of encapsulation, which was originally introduced in object-oriented systems to facilitate modular design, can now be employed for purposes of security. The conceptual clarity and simplicity of the model translates into simplicity of design of security mechanisms.

An informal view of the model describes the system as consisting of objects that are assigned unique security classifications. Objects can communicate among themselves *only* by means of sending messages. Messages cannot, however, flow directly from one object to another. Instead, they have to be screened by a security module that decides how to handle any given message based on the security classifications of the sender and the intended receiver as well as some additional information. The rulings issued by this module embody the security policy.

Section 2 is reproduced from [4]. In it, we define our formal object-oriented data model. Section 3 is an exposition of the security model rendered in such a fashion as to address implementation concerns more explicitly than it was done in [4]. Section 4 explains why our model is conceptually easy to implement in object-oriented environments. Section 5 discusses some alternatives for implementation of object-oriented databases based on our model, particularly the issue of which elements of the system have to be trusted. Section 6 addresses some requirements placed by our model on the implementation of the object layer. Finally, Section 7 ends this report with some concluding remarks and observations.

---

else to access the object's state, it is necessary to send a message to that object.

2. Object-Oriented Data Model.

An object-oriented database is a collection of *objects* communicating via *messages*. Each object consists of a unique identifier, a set of *attributes* and a set of *methods*, the latter being essentially pieces of code. Each attribute has a *value*, which can change over time.

An object can invoke one of its methods in response to a message received from another object. A method invocation can, in turn, (1) directly access an attribute belonging to the object (read or change its value); (2) invoke other methods belonging to the object; (3) send a message to another object; or (4) create a new object.

There is a special type of object, called *user* object. A user object represents a user session with the system. User objects differ from regular objects in that, in addition to being able to invoke methods in response to messages, they can also invoke methods spontaneously.† User object can be created only by the system, at the login time.

Let us formalize now the central elements of the object-oriented data model. We postulate a finite set of domains $D_1, D_2, ..., D_n$. Let D be the union of the domains augmented with a special element *nil*, i.e., $D = D_1 \cup D_2 \cup \cdots \cup D_n \cup \{nil\}$. Every element of D is referred to as a *primitive object*. Let A be a set of symbols called *attribute*

† The notion of spontaneous method invocation may seem rather arbitrary at first. It is, however, necessary in order to avoid running into a version of the chicken-and-egg paradox. Namely, if a message can be sent only through a method invocation (see property (3) of method invocations) and if a method can be activated only by a message received from another object, then how does any processing in such a system ever get initiated? (One has to insist that either the egg or the chicken come first.) In reality, we want a user to be able to initiate a system activity, e.g., by typing a string of characters on the keyboard. This would serve as a signal for the corresponding user object to initiate a method. We choose to think of this as a "spontaneous" initiation, because the keyboard and any signals that it sends are external to our model.

names, I a set of *identifiers*, and M a set of finite strings of code called *methods*. Let V be a set of *values* defined as follows: $V = D \cup I \cup 2^I$. That is, a value is either a primitive object or an identifier or a set of identifiers.

Definition 1. An *object* is either a primitive object or a quadruple $o = (i, a, v, \mu)$ such that $i \in I$, $a = (a_1, ..., a_k)$ where $a_j \in A$ for all j $(1 \le j \le k)$, $v = (v_1, ..., v_k)$ where $v_j \in V$ for all j $(1 \le j \le k)$, and $\mu \subset M$. □

Definition 1 states that an object is defined by its identifier, an ordered set of attribute names, an ordered set of corresponding values, and a set of methods. We assume that every object has a unique identifier, i.e., for any two objects $o_s = (i_s, a_s, v_s, \mu_s)$ and $o_t = (i_t, a_t, v_t, \mu_t)$, $o_s = o_t$ iff $i_s = i_t$. The uniqueness of object identity is commonly considered a fundamental property of object-oriented systems [5].

We will use the following notation in the foregoing discussion. Let $o_s = (i_s, a_s, v_s, \mu_s)$ be an object. Then $i(o_s)$ denotes the object identifier, i_s; $a(o_s)$ denotes the list of attributes, a_s; $v(o_s)$ denotes the list of attribute values, v_s, and $\mu(o_s)$ denotes the object's set of methods, μ_s.

Definition 2. A *message* is a triple $g = (h, p, r)$ where h is the *message name*, $p = (p_1, ..., p_k)$, $k \ge 0$, is an ordered set (list) of values called the *message parameters*, and r is the *return value*. □

Similarly to the notation used for objects, we let $h(g)$, $p(g)$, and $r(g)$ denote the name, the parameter list, and the return value of message g respectively.

An object sends a message by invoking a system primitive $SEND(g, i)$ where i is the identifier of the receiver object. The value $r(g)$ is computed by the method activated

in the receiver upon the arrival of g there and returned to the sender.[†]

In the literature on object-oriented data models, the response to a message is often defined as a return *object*. We use the notion of variable instead, in order to avoid dealing with the issue of accessing the state of the return object, which, in compliance with the property of encapsulation, would have to be done via message sending (unless the return object is a primitive object). Sending messages to an object that has been returned in response to another message seems conceptually cumbersome. The notion of variable, on the other hand, implies direct accessibility without the need to resort to message sending. Of course, return variables should be allowed to have arbitrarily complex structure if we do not want to lose the modeling power associated with objects.

Definition 3. The *interface* f_o of object o is a function $f_o: H \rightarrow \mu(o) \cup \{void\}$ where H is a set of all possible message names. □

The interface of object o determines which messages o responds to. Those are the messages whose names, h, are such that $f_o(h) \neq void$. If $f_o(h) = void$, o does not respond to messages whose name is h. Moreover, the interface determines which particular method, out of the set of methods, $\mu(o)$, defined for object o, is to be invoked, depending on the name of the given message.

We have defined methods as strings of code. Now we are in a position to give a more formal definition of methods.

[†] As we shall see in the next section, sometimes the security component of the system will have to interfere in the matter of computing $r(g)$.

Definition 4. Let o be an object. A *method* m defined for o ($m \in \mu$) is a function $m: P \rightarrow 2^{a(o) \times V} \times 2^{G \times I} \times V$ where P is a set of all possible parameter lists. □

Definition 4 states that a method maps a list of parameters into a triple. The first element of the triple is a set (possibly empty) of attribute-name—attribute-value pairs where the names are drawn from the set of the object's attribute names. The second element is a set (possibly empty) of message—identifier pairs. The third element is a value.

In response to a message $g = (h, p, r)$, an object o invokes a method $m \in \mu(o)$ such that $m = f_o(h)$ (we assume that $f_o(h) \neq void$). Then, the value $m(p)$ is computed (this corresponds to executing the method's code with the argument list p). The computation results in $m(p) = (\{(a_1, v_1), ..., (a_s, v_s)\}, \{(g_1, i_1), ..., (g_t, i_t)\}, v_j\}$. The semantics of this are as follows. Attributes $a_1, ..., a_s$ of o are updated with new values $v_1, ..., v_s$ respectively; messages $g_1, ..., g_t$ are sent to the objects with identifiers $i_1, ..., i_t$ respectively; and v_j is returned to the sender of g. Note that for some k we could have $i_k = \iota(o)$, i.e., an object can send a message to itself. This, for instance, can serve as a mechanism for invoking other methods within the same object.

Objects are used to model real-world entities.† This is done by associating properties, or facets, of an entity with attributes of the corresponding object. The attribute values are, then, instantiations of those properties. For instance, a country can be represented in a geographic object-oriented database by an object o where $a(o) =$ (COUNTRY_NAME, POPULATION, CAPITAL, NATIONAL_FLAG, FORM_OF_GOVERNMENT) and $v(o) =$ (''Albania'', 117, $i(o_1)$, $i(o_2)$, $i(o_3)$). The

† As we will see later, a single entity may be modeled by more than one object.

values of the first and second attributes are a string and an integer, respectively; the values of the rest of the attributes are references to other objects that, in turn, describe the capital, the national flag, and the form of government of the nation of Albania.

Note that an object's methods, unlike its attributes, do not have counterparts in the real-world entity modeled by the object. The purpose of methods is quite different. It is to provide support for basic database functionality such as querying and updating objects.

A realistic object-oriented model should also contain the notion of constraints. For instance, an attribute of an object may be allowed to assume values only from a restricted subset of domains or object identifiers. To simplify the exposition, we choose to disregard the issue of constraints in this report. However, it should be a simple matter to incorporate this notion in our security—data model.

3. Object-Oriented Security Model.

The system consists of a set O of objects (see Definition 1) and a partially ordered set S of *security levels* with ordering relation $<$. A level $S_i \in S$ is said to be *dominated* by another level $S_j \in S$, this being denoted by $S_i \leq S_j$, if $i = j$ or $S_i < S_j$. For two levels S_i and S_j that are unordered by $<$, we write $S_i <> S_j$.

There is a total function $L: O \rightarrow S$, called *security classification* function, i.e., for every $o \in O$, $L(o) \in S$. In other words, every object has a unique security level associated with it.

3.1. Characterization of Information Flows.

Information can *legally* flow from an object o_j to an object o_k if and only if $L(o_j) \leq L(o_k)$. All other information flows are considered *illegal*.

In [4] we identified basic types of information flow. They are as follows: *forward*, *backward*, *transitive*, and *indirect* flows. The forward flow is carried by a message in its parameter list. The backward flow is associated with a message's return value. The transitive flow is the net effect of several forward or backward flows along a chain of objects. The indirect flow occurs between two objects that do not exchange messages with each other but, instead, exchange messages with a third object; the latter is not, however, directly affected by the flow (unlike the case of the transitive flow).

As argued in [4], for an object to acquire information, the values of some of its attributes must be changed. Therefore, one can prevent an illegal forward information flow by making sure that no such changes occur as a result of a method invocation in response to a message. Similarly, an illegal backward flow is prevented by returning *nil* in response to a message. Note that *nil* is also returned when the message is undeliverable for whatever reason (e.g., the target object does not exist). Next, by definition, if no illegal forward or backward flows are allowed, no illegal transitive flow can occur either. Finally, to prevent illegal indirect information flows, one must ensure that a method is prevented from updating any local attributes inside the object if the method is invoked in response to a message from another object that is at a higher or unrelated security level.

3.2. The Message Filtering Algorithm

To control all the types of information flow described above, the message filtering algorithm was introduced in [4]. The idea that information flow be controlled by controlling the flow of messages requires that all basic object activity — such as access to internal attributes, new object creation, and invocation of local methods — be implemented by allowing an object to send messages to itself.† These are built-in, or system-defined, messages. This means that a response to such a message is carried out directly by the systems, according to some pre-defined semantics, rather than by the invocation of a user-defined method.

For easy reference, we reproduce here the message filtering algorithm of [4] (see Figure 1).

In Figure 1, $g = (h, p, r)$ is a message. Objects $o_1 = (i_1, a_1, v_1, \mu_1)$ and $o_2 = (i_2, a_2, v_2, \mu_2)$ are the sender and the receiver of g respectively. The method invocation in o_1 responsible for sending g is denoted t_1. Finally, t_2 is the invocation of the method $f_{o_2}(h)$ in o_2 after receiving g. Every method invocation t has a *status* $s(t)$. The status is either U (unrestricted) or R (restricted). The default is U.

The message filtering algorithm is the core of our security model. The effect of making this security-modeling choice is that all information-transfer-related activity is rendered explicit in the form of message sending. This has a direct relation to the question of implementation, for such activity is now subject to monitoring by a security module called the *message filter*, whose functionality is based on the message filtering

† There are existing object-oriented database systems that, in fact, use this kind of implementation, e.g., GemStone.

algorithm.

4. Algorithmic Nature of the Model.

Our security model has one distinctive feature that makes the question of implementation relatively easy to address. That is the fact that virtually the entire model can be expressed as one simple algorithm. When coded and implanted into the system that supports objects, this algorithm becomes a security module that we refer to as the *message filter*. The role of this module is to act as an interceptor for every message originated by any object and decide how to process that message. Figure 2 shows the interaction among objects as taking place with mediation on the part of the message filter.

Thus, the model already contains the prescription for its own implementation. This fact is due to the model's *algorithmic* nature. Note that this is in contrast with other existing database security models, which are *constraint-based*. Models of the latter kind require some additional work on mechanisms for enforcing those constraints before the actual implementation can take place. In addition, those models probably could not be implemented as a single module because the constraint checking would have to be done in a number of different logical places.

5. Placement of Trust, or What to Rely on.

In this section, we discuss what is perhaps one of the central implementation questions for any secure system: which components of the system need to be trusted.

CASE A: $o_1 \neq o_2$

(1) **if** $L(o_1) = L(o_2)$
let g **pass**; $s(t_2) \leftarrow s(t_1)$

(2) **if** $L(o_1) <> L(o_2)$
block g

(3) **if** $L(o_1) < L(o_2)$
let g **pass**; $r \leftarrow nil$; $s(t_2) \leftarrow s(t_1)$

(4) **if** $L(o_2) < L(o_1)$
let g **pass**; $s(t_2) \leftarrow R$

CASE B: $o_1 = o_2$

(1) **if** $h = WRITE$

(1.a) **if** $s(t_1) = U$
let g **pass**

(1.b) **if** $s(t_1) = R$
block g

(2) **if** $h = READ$
let g **pass**

(3) **if** $g = (CREATE, \{v_1, v_k, ..., S_j\}$

(3.a) **if** $s(t_1) = U$ **and** $(S_j < L(o_1)$ **or** $S_j <> L(o_1)$
let g **block**;

(3.b) **if** $s(t_1) = R$
block g

(3.c) **if** $s(t_1) = U$ **and** $L(o_1) \leq S_j$
let g **pass**

(4) **if** $h = INVOKE$
let g **pass**; $s(t_2) \leftarrow s(t_1)$

Figure 1. The Message Filtering Algorithm.

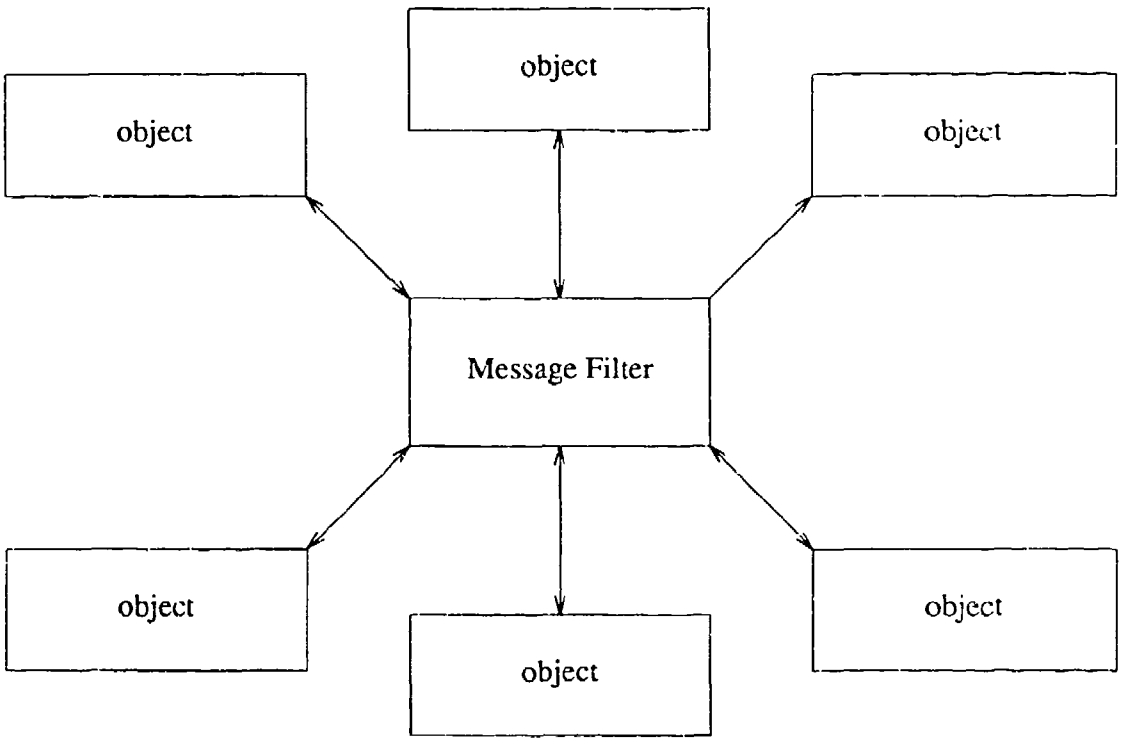

Figure 2.

5.1. The TDI Approach: Complementing the Reference Monitor with the Message Filter.

One approach to this question is dictated by the traditional notion of TDI (Trusted Database Implementation). This approach relies on the underlying security kernel of the operating system. The kernel is the trusted component that implements the mandatory access control in the context of multilevel security. The role of the database security model in such a setting is to provide a high level (data-model level) interpretation of the security policy for database users.

In accordance with this approach, the central element of our model — the message

filter — does not have to be trusted, since the actual security enforcement is done at the operating system level by the reference monitor. The latter checks all data access requests submitted to the operating system by user processes to make sure that no violations of the access policy occur.

The TDI approach is the widely used way of dealing with database security requirements. Examples are the SeaView model [1,3] and the security model for object-oriented applications developed at SRI [6]. Our security model can be just as easily implemented using the TDI approach as the above two. In this implementation, the message filter would provide the database interpretation for the actual enforcement of the security policy by the reference monitor. The reference monitor would have to be trusted; the message filter would not.

5.2. The High Level Approach: Replacing the Reference Monitor with the Message Filter.

One of the advantages of our model, however, is that it also makes possible an alternative method of implementing the system and placing trust among its components. This new method has potential advantages over the more traditional one. Below, we discuss these advantages following the discussion of why the alternative method is feasible.

But first, it would be useful to compare the functionality and role of a message filter with those of a reference monitor. There is an interesting parallel between the two. The function of both is to control information-transfer activity in the system, based on the presumption that (1) *all* such activity is carried out using a predefined set of primitives,

and (2) an instance of usage of a primitive can always be detected and acted upon appropriately by the security module (a message filter or reference monitor) with the objective of preventing illegal information flows. In the case of reference monitor, the primitives are data access requests submitted to the operating system by an active process (usually in the form of system calls). In the case of message filter, the primitives are messages sent by method invocations.

The majority of actual computer systems today are implemented using data access calls. That is why the notion of reference monitor is both natural and effective.

Imagine, however, that messages replaced data access calls as the basic primitive. Then it would be just as natural and easy to guard against illegal information transfer using a message filter as it was using a reference monitor in the case of data access calls. Such an assumption is not far fetched at all since systems exist built on the message sending primitive [8].

Thus, when messages replace data access calls as primitives, a message filter should replace a reference monitor as the enforcer of security. The placement of trust, then, shifts from the latter to the former, i.e., the former must now be trusted and the latter disappears completely. Such state of affairs seems quite favorable to our model of database security as well as to the problem of secure object-oriented databases in general. The reason for this is that there is no longer a separation between security enforcement and its interpretation: they have become one and the same. The message filter does not rely any more on any underlying security kernel but rather enforces security directly by regulating the flow of messages among objects. This situation, we believe, is more

effective and reliable than a two-tiered implementation of the kind of TDI.

6. Object Security and Object Implementation.

In order for our approach to object security to be effective, the implementation of objects themselves must satisfy certain requirements. In other words, our security model is not indifferent to how specifically the object layer is implemented.

This subject has been already touched at several points earlier in this report and especially in [4]. In this section, the subject is brought into focus and given a brief unified treatment.

All such implementation requirements imposed by the security model can be expressed in the following general form: *Every object activity related to information transfer must be implemented by message-passing*. This means that in addition to interaction among objects, message-passing must also be used for the following:

(1) reading of local attributes by an object,

(2) updating of local attributes,

(3) invocation of local methods, and

(4) inheritance (both class-instance and class-subclass).

When the above requirements are satisfied, all information activity in the system can be checked directly by the message filter. As was mentioned in [4], some existing object-oriented systems indeed implement access to local attributes and local method invocation by having an object send a primitive message to itself. The situation is analogous to a system where processes have to issue system calls in order to access data or

activate a procedure. Therefore, there is nothing unusual about these requirements.

Implementing the inheritance mechanism by means of message passing is not a new concept either (e.g., see [7]). Its essence is in redirecting of a message sent to an object such that the method called for is not physically located in the object. For example, all the methods defined for a class of objects are stored within the class object for that class. Therefore, when a message is sent to an instance of that class, it will be redirected to the class object so that the needed method can be invoked there. Subsequently, if the method invocation requires access to the instance object's attributes, messages will be sent to the latter for that purpose. A further redirection of messages can occur if the method in question is inherited from a superclass rather than defined locally for the class.

Thus, the requirements placed by our security model on the implementation of the object layer are by no means unreasonable. However, the very fact that there are some requirements will preclude using our model with just any object-oriented database. We do not consider this a detriment, though, for the following reason. Our interest is in constructing an effective and comprehensive approach to security for object-oriented databases, not just coming up with minimally functional mechanisms that can be fitted on top of any existing database.

7. Conclusions.

We have attempted to present here an integrated treatment of the questions of security modeling and security implementation in the context of object-oriented databases. Such methodology seems particularly appropriate because the security model that we are

using is, by its very nature, both abstract and implementation specific.

Perhaps, the most important novel idea found in this report is that of supplementing, or perhaps even replacing, the traditional reference monitor with the message filter, which seems to be a natural choice in object-oriented databases.

This report along with its companion report on the object-oriented data and security models [4] represent an initial step in what we hope will be a long-term research effort towards designing and building multilevel secure object-oriented databases.

References

1. Teresa F. Lunt , Roger R. Schell, William R. Shockley, Mark Heckman, and Dan Warren, "A near-term design for the SeaView multilevel database system," *Proc. Symp. on Security and Privacy*, pp. 234-244, April 1988.

2. D. E. Bell and L. J. LaPadula, "Secure computer systems: Unified exposition and multics interpretation," The Mitre Corp., March 1976.

3. Dorothy E. Denning, Teresa F. Lunt , Roger R. Schell, William R. Shockley, and Mark Heckman, "The SeaView security model," *Proc. Symp. on Security and Privacy*, pp. 218-233, April 1988.

4. Sushil Jajodia and Boris Kogan, "Integrating an object-oriented data model with multilevel security," Report prepared for the Rome Air Development Center, December 1989.

5. Setrag N. Khoshafian and George P. Copeland, "Object Identity," *Proc. Conf. on Object-Oriented Programming: Systems, Languages, and Applications*, pp. 406-416,

1986.

6. Teresa F. Lunt and Jonathan K. Millen, "Secure knowledge-based systems," Interim Technical Report, Computer Science Laboratory, SRI International, August 1989.

7. Naftaly H. Minsky and David Rozenshtein, "A law-based approach to object-oriented programming," *Proc. Conf. on Object-Oriented Programming: Systems, Languages, and Applications*, pp. 482-493, October 1987.

8. Andrew S. Tanenbaum and Robert van Renesse, "Distributed Operating System," *ACM Computing Surveys*, vol. 17, no. 4, pp. 419-470, December 1985.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.